UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Secure Communications: IPSec and TLS

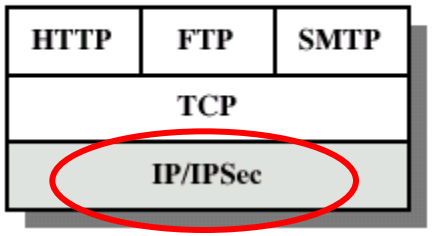
Luca Veltri

(mail.to: luca.veltri@unipr.it)

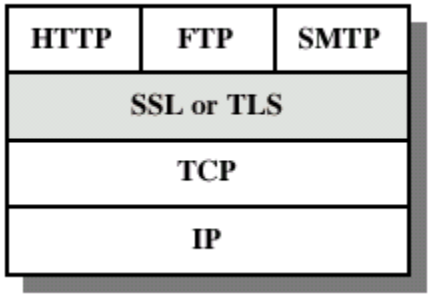
Course of Cybersecurity, 2022/2023

http://netsec.unipr.it/veltri

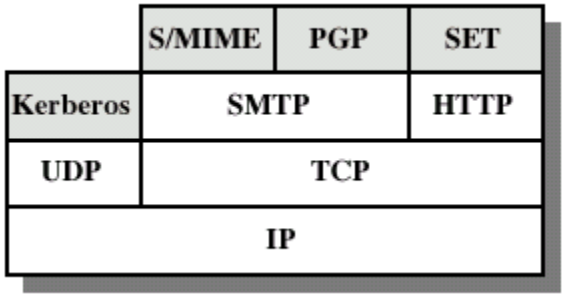
Security within IP stack protocols



Network level



transport/session level



Application level

IPSec

IPSec Introduction

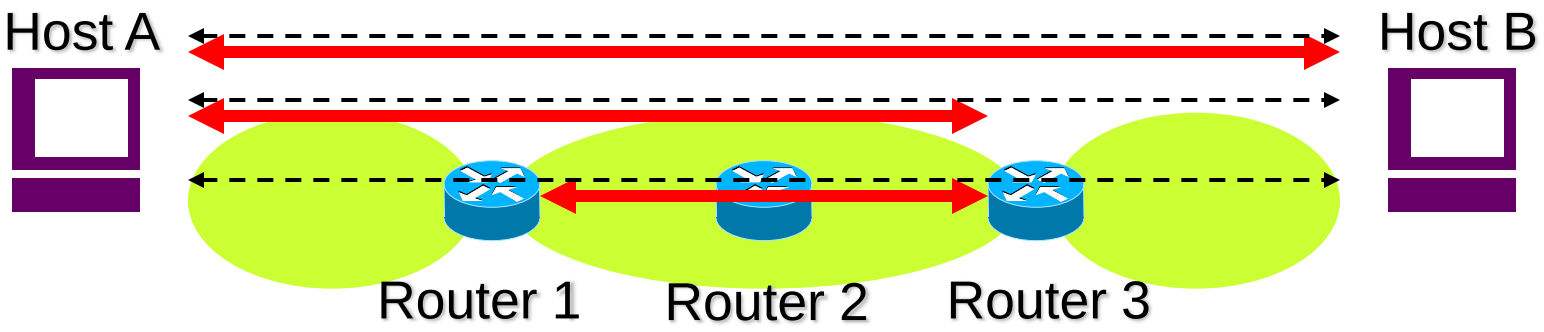
- IPSec, as defined in RFC 4301 and successive standards, is a framework for providing interoperable, high quality, cryptographically-based security for IPv4 and IPv6
  - **offering protection in a standard fashion for all protocols that may be carried over IP (including IP itself)**
- The set of security services offered includes
  - **access control**
  - **connectionless data integrity, data origin authentication**
  - **detection and rejection of replays**
  - **confidentiality**
  - **and limited traffic flow confidentiality**

IPSec Introduction (cont.)

- An IPSec implementation operates in a host, or as a security gateway (SG), affording protection to IP traffic
  - **a security gateway is an intermediate system implementing IPSec, e.g., a firewall or router that has been IPSec-enabled**
- IPSec does not adversely affect users, hosts, and other Internet components that do not employ IPSec for traffic protection
  - **transparent for intermediate nodes that do not implement it**
  - **transparent for end nodes (host) that do not implement it, when implemented by intermediate nodes**
  - **transparent for end applications that are not aware of it**
- Hence, IPSec is very useful in presence of legacy hosts or applications
- One IPSec implementation may protect traffic from/to several hosts and applications

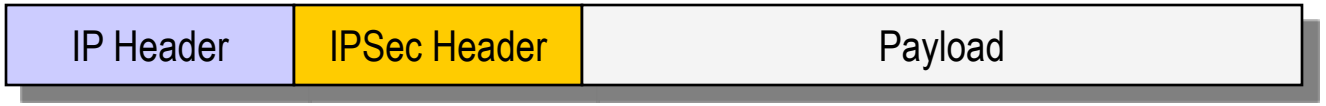
IPSec Introduction (cont.)

- IPsec can be used to protect one or more "paths"
  - **between a pair of hosts (host-to-host)**
    - end-to-end, alternative to transport or application level security solutions
  - **between a pair of security gateways (router-to-router)**
    - often referred to as VPN/IPSec
  - **between a security gateway and a host (host-to-router)**
    - often referred to as “road-warrior”

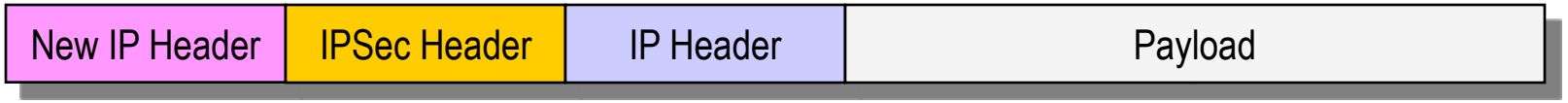


Transport & Tunnel Modes

- IPSec supports two modes of use:
  - **transport mode**
    - IPSec provide protection primarily next layer protocols
    - The source and destination are the IPSec endpoints

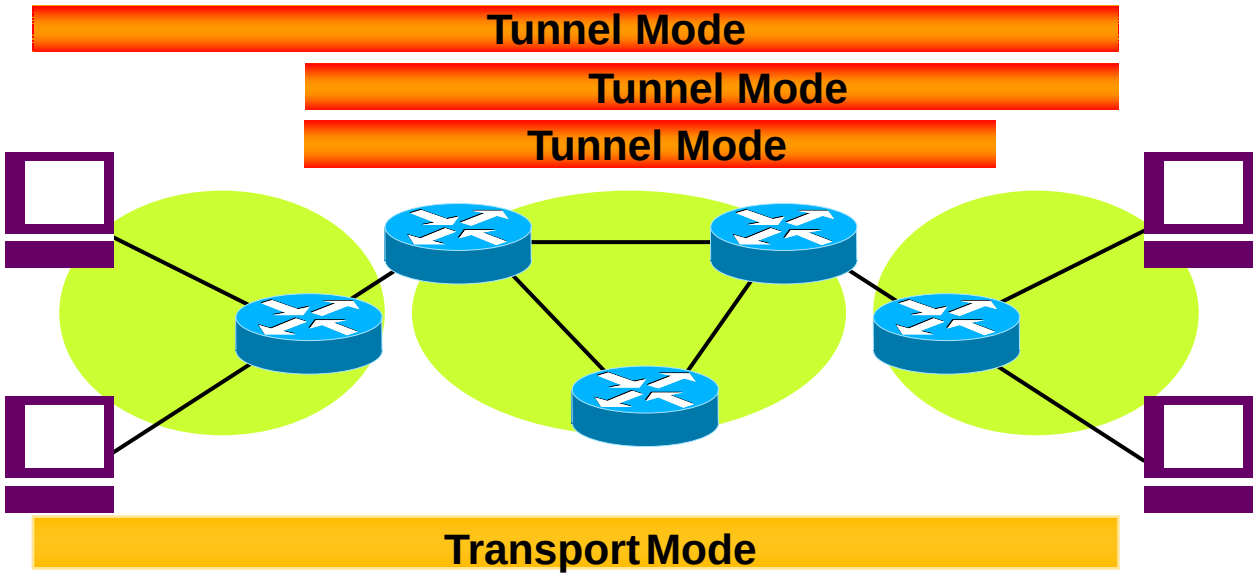


  - **tunnel mode**
    - IPSec is applied to tunneled IP packets
    - The outer IP header Source and Destination Addresses identify the "endpoints" of the tunnel (the encapsulator and decapsulator)
    - The inner IP header Source and Destination Addresses identify the original sender and recipient of the datagram
    - The inner IP header is not changed except for TTL and the DS fields



Transport & Tunnel Modes (cont.)

- Transport mode can be used only when source and destination addresses coincide with the addresses of the IPSec endpoints



IPSec protocols

- Two protocols have been originally specified for securing IPSec communications
  - **the Authentication Header (AH)**
    - originally designed to offer integrity and data origin authentication without confidentiality, with optional (at the discretion of the receiver) anti-replay features
    - support is optional
    - rarely used
  - **Encapsulating Security Payload (ESP)**
    - offers integrity, data origin authentication, and data confidentiality
    - there are also provisions for limited traffic flow confidentiality, i.e., provisions for concealing packet length, and for facilitating efficient generation and discard of dummy packets
      - this capability is likely to be effective primarily in virtual private network (VPN) and overlay network contexts

IPSec protocols (cont.)

- support for AH is optional and experience has shown that there are very few contexts in which ESP cannot provide the requisite security services
  - **ESP can be also used to provide only integrity, without confidentiality, making it comparable to AH in most contexts**
  - **as a result, AH is rarely used**
- Because IPSec security services require the use of cryptographic keys, IPSec relies on a separate mechanism for putting these keys in place
  - **IPSec specifies IKE (Internet Key Exchange) as public-key based key management mechanism**
  - **other automated key distribution techniques may also be used**

Cryptographic algorithm independency

- IPsec security protocols (ESP, AH, and IKE) are designed to be cryptographic algorithm independent
  - **this modularity permits selection of different sets of cryptographic algorithms as appropriate, without affecting the other parts of the implementation**
  - **different users may select different sets of cryptographic algorithms**
  - **to facilitate interoperability in the global Internet, a set of default cryptographic algorithms for use with AH and ESP, and a set of mandatory-to-implement algorithms for IKEv2, are specified**

Security Association (SA)

- A Security Association (SA) is a simplex logical "connection" that affords security services to the traffic carried by it
  - **specifies algorithms, key material, and parameters to operate the AH and/or ESP operations**
  - **a major function of IKE is the establishment and maintenance of SAs**
  - **to secure typical, bi-directional communication between two IPsec-enabled systems, a pair of SAs (one in each direction) is required**

Encapsulating Security Payload (ESP)

- Encapsulating Security Payload (ESP) (RFC 4303) is designed to provide a mix of security services
  - **confidentiality**
  - **data origin authentication and connectionless integrity**
  - **anti-replay service (a form of partial sequence integrity)**
  - **(limited) traffic flow confidentiality**
    - by adding padding bytes after the end of the payload data
- Confidentiality and integrity can be offered independently
  - **only data authentication**
    - ESP is simpler and faster to process than AH
  - **only encryption**
    - this will provide defense only against passive attackers
      - using encryption without a strong integrity mechanism may render the confidentiality service insecure against some forms of active attacks
- ESP typically will employ both services

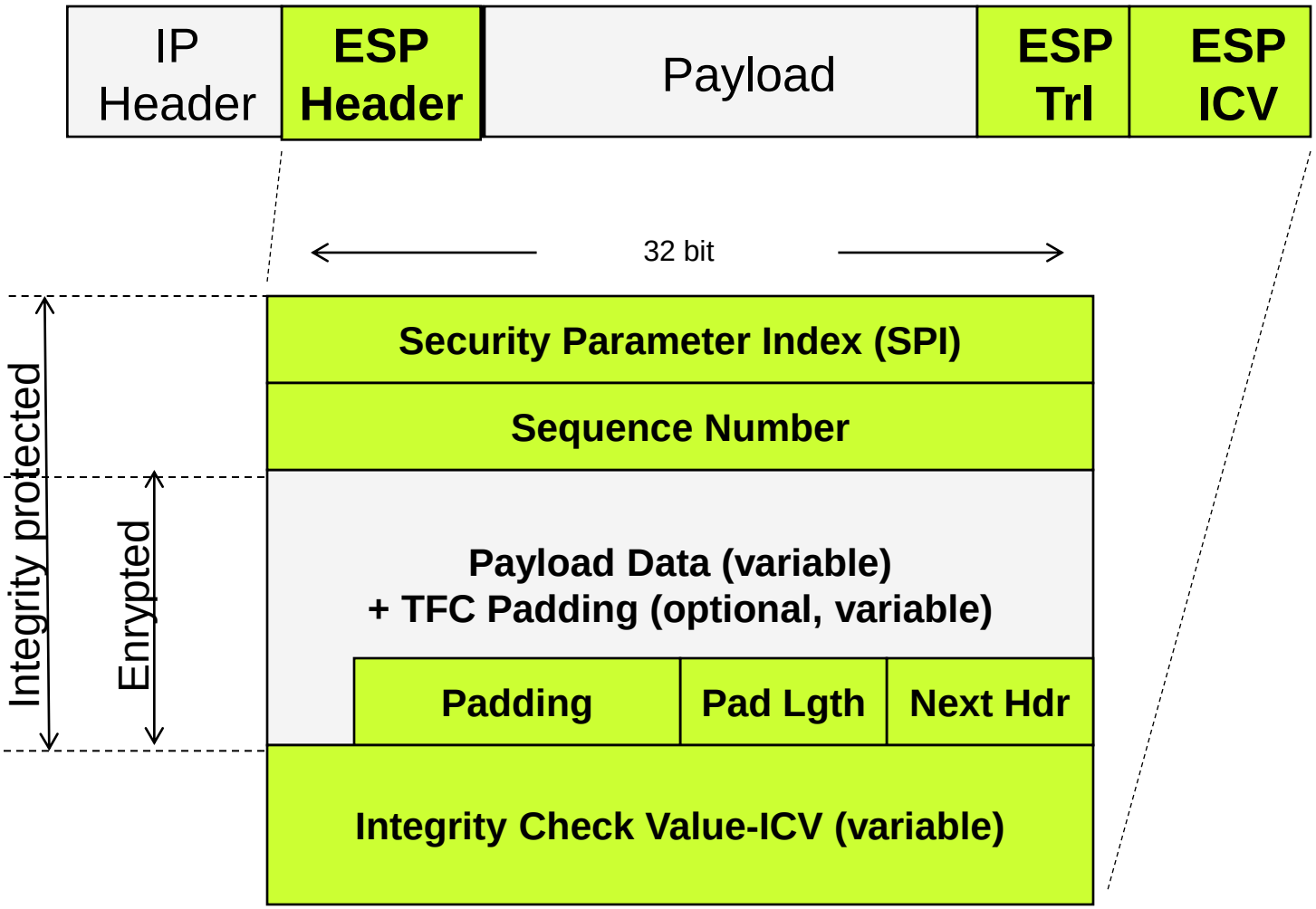
ESP (cont.)

- The anti-replay service may be used only if the integrity service is provided
  - **the selection of this service is solely at the discretion of the receiver and thus need not be negotiated**
- The traffic flow confidentiality (TFC) service generally is effective only if
  - **ESP is employed in a fashion that conceals the ultimate source and destination addresses of correspondents, e.g., in tunnel mode between SEGs, and**
  - **sufficient traffic flows between IPsec peers (either naturally or as a result of generation of masking traffic) to conceal the characteristics of specific, individual subscriber traffic flows**

ESP (cont.)

- ESP may be applied alone, in combination with AH, or in a nested fashion
- The ESP header is inserted after the IP header and before the next layer protocol header (transport mode) or before an encapsulated IP header (tunnel mode)

ESP Packet Format



ESP Packet Format (cont.)

- Security Parameter Index (SPI)
  - **an arbitrary 32-bit value that is used by a receiver to identify the SA to which an incoming packet is bound**
- Sequence number
  - **unsigned 32-bit field contains a counter value that increases by one for each packet sent, i.e., a per-SA packet sequence number**
- Payload data
  - **variable-length field containing data (from the original IP packet) described by the Next Header field**
    - If the algorithm used to encrypt the payload requires cryptographic synchronization data, e.g., an Initialization Vector (IV), then this data is carried explicitly in the Payload field
- Traffic Flow Confidentiality (TFC) Padding
  - **added only if the Payload Data field contains a specification of the length of the IP datagram**
    - this is always true in tunnel mode, and may be true in transport mode depending on whether the next layer protocol (e.g., IP, UDP, ICMP) contains explicit length information

ESP Packet Format (cont.)

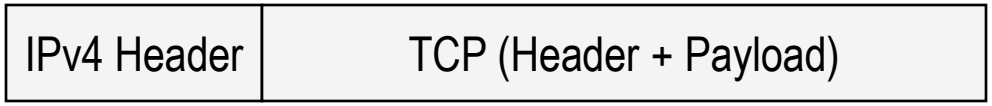
- Padding (for Encryption)
  - **0 to 255 bytes of padding**
    - If the encryption algorithm requires the plaintext to be a multiple of some number of bytes, e.g., the block size of a block cipher
    - or to ensure that the resulting ciphertext terminates on a 4-byte boundary
    - note: a separate mechanism should be used for TFC (if required)
- Pad length
  - **the number of pad bytes in the Padding field**
- Next header
  - **8-bit field that identifies the type of data contained in the Payload Data field, e.g., an IPv4 or IPv6 packet, or a next layer header and data**
- Integrity Check Value (ICV)
  - **(optional) variable-length field computed over the ESP header, Payload, and ESP trailer fields**

ESP Packet Format (cont.)

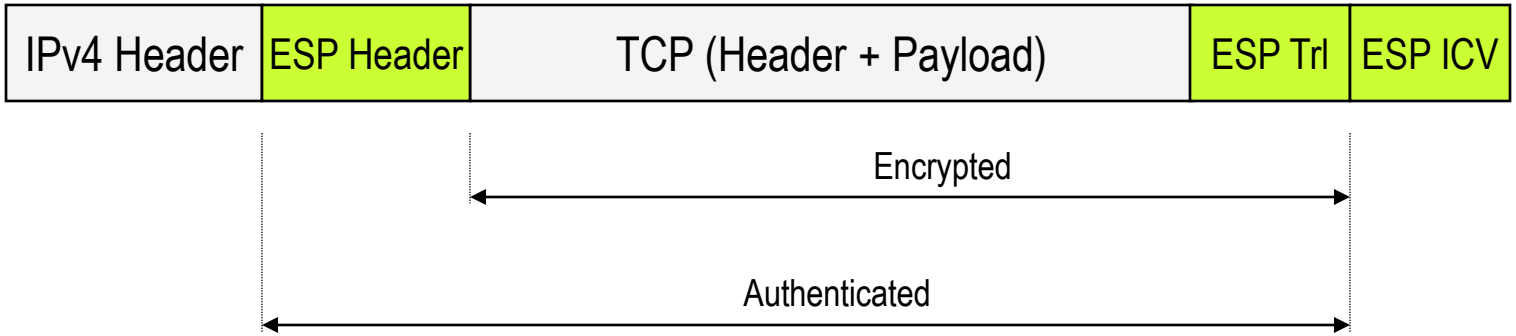
- If the confidentiality service is selected, the encrypted data consists of the Payload Data and the ESP trailer
- If the integrity service is selected, the integrity computation encompasses the SPI, Sequence Number, Payload Data, and the ESP trailer

ESP - Transport Mode

- In transport mode, ESP is inserted after the IP header and before a next layer protocol, e.g., TCP, UDP, ICMP, etc.
  - **in IPv4, this translates to placing ESP after the IP header (and any options that it contains), but before the next layer protocol**
- Example of IPv4 packet before applying ESP

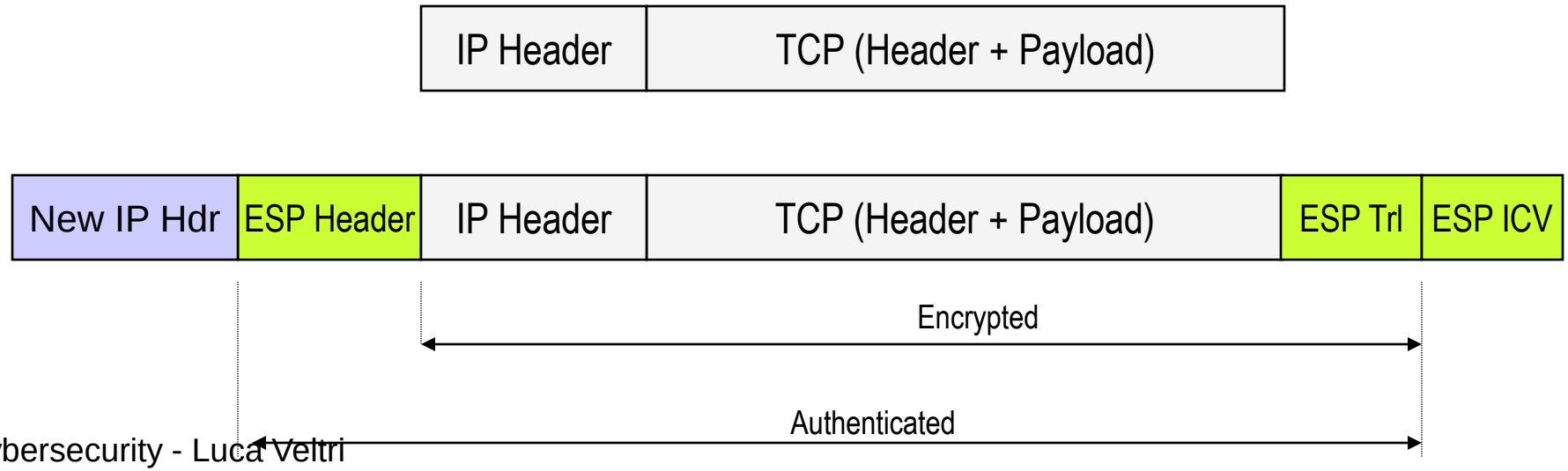


- The same packet after applying ESP in transport mode



ESP - Tunnel Mode

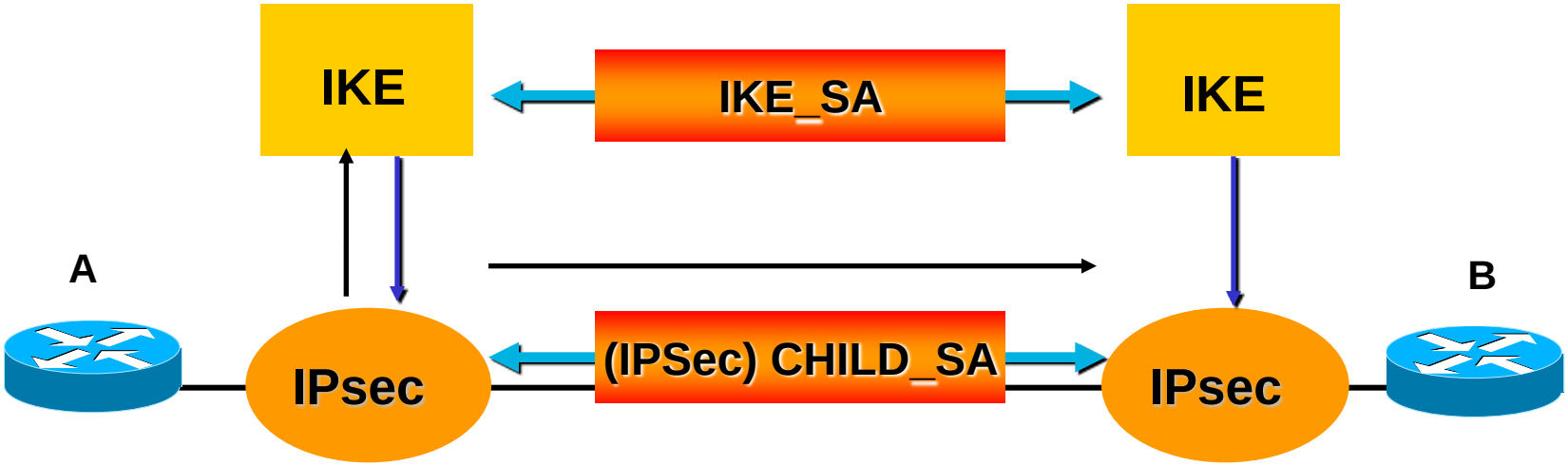
- In tunnel mode, the "inner" IP header carries the ultimate (IP) source and destination addresses, while an "outer" IP header contains the addresses of the IPsec "peers", e.g., addresses of SEGs
  - **mixed inner and outer IP versions are allowed, i.e., IPv6 over IPv4 and IPv4 over IPv6**
  - **ESP protects the entire inner IP packet, including the entire inner IP header**



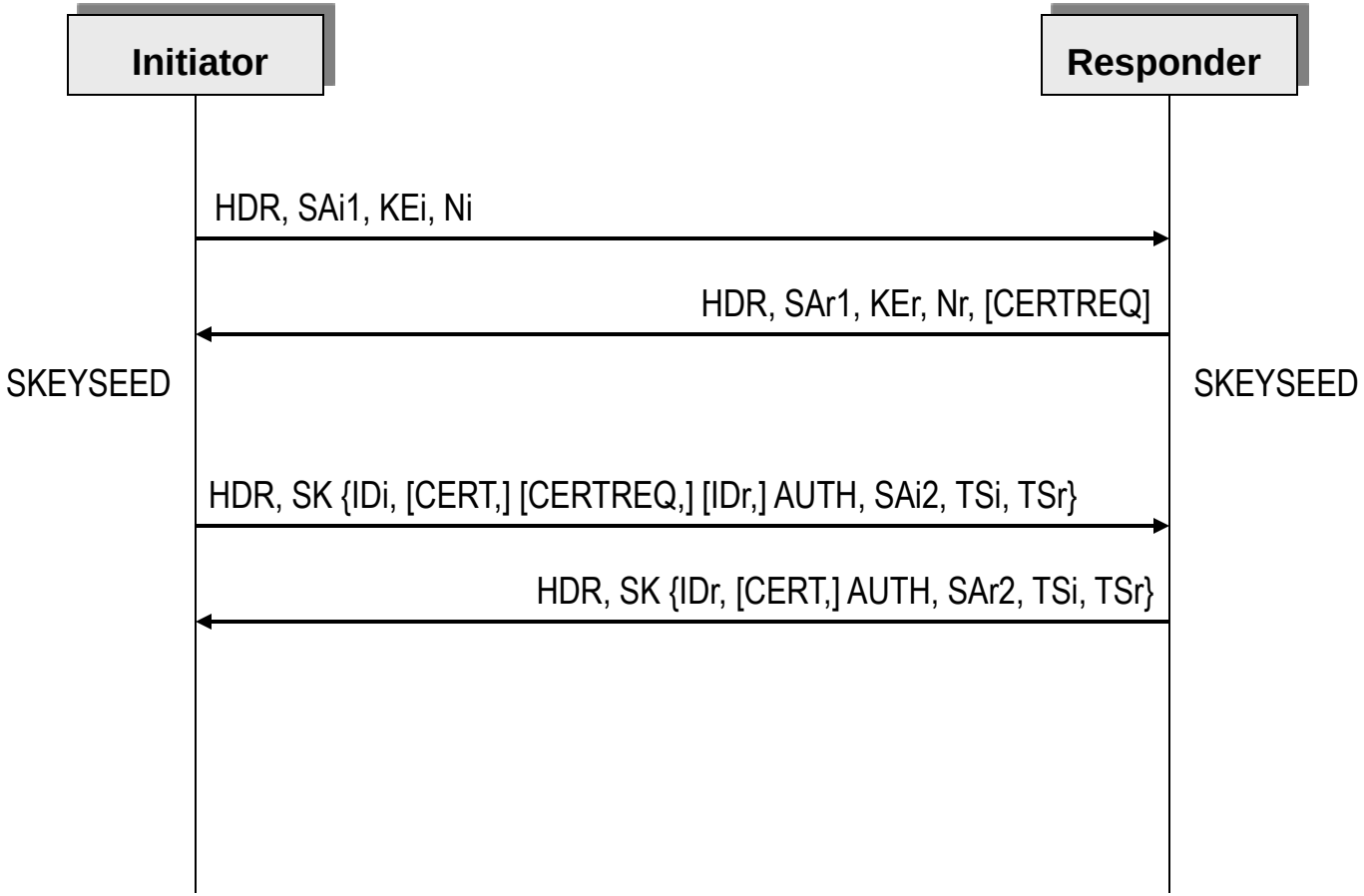
Internet Key Exchange (IKEv2) Protocol

- IP Security (IPSec) provides confidentiality, data integrity, access control, and data source authentication to IP datagrams
- These services are provided by maintaining shared state (the Security Association) between the source and the destination
  - **this state defines the specific services provided to the datagram, which cryptographic algorithms will be used to provide the services, and the keys used as input to the cryptographic algorithms, etc.**
  - **establishing this state in a manual fashion does not scale well**
- Internet Key Exchange (IKEv2) performs mutual authentication between two parties and establishes an IKE SA (IKE_SA) that includes shared secret information that can be used to efficiently establish SAs for ESP and/or AH (CHILD_SAs)

IKE_SA and CHILD_SAs



Initial Exchanges: IKE_SA_INIT and IKE_AUTH



IKE message notation

- Notation:

AUTH	**Authentication**
CERT	**Certificate**
CERTREQ	**Certificate Request**
CP	**Configuration**
D	**Delete**
EAP	**Extensible Authentication**
HDR	**IKE header (not a payload)**
IDi	**Identification - Initiator**
IDr	**Identification - Responder**
KE	**Key Exchange**
Ni, Nr	**Nonce**
N	**Notify**
SA	**Security Association**
SK	**Encrypted and Authenticated**
TSi	**Traffic Selector - Initiator**
TSr	**Traffic Selector - Responder**
V	**Vendor ID**

Initial Exchanges: IKE_SA_INIT and IKE_AUTH (cont.)

- IKE always begins with IKE_SA_INIT and IKE_AUTH exchanges (known in IKEv1 as Phase 1)
  - **HDR contains the SPIs, version numbers, and flags**
  - **SA states the cryptographic algorithms for the IKE_SA**
  - **The KE contains the Diffie-Hellman value**
  - **Ni and Nr are nonce values**
  - **SKEYSEED is the DH secret from which all keys are derived for that IKE_SA**
    - all but the headers of all the messages that follow are encrypted and integrity protected by SK{·}
    - SKEYSEED is calculated as: = PRF(Ni | Nr, g^ir)
    - from SKEYSEED further secrets are then generated:

      SK_d | SK_ai | SK_ar | SK_ei | SK_er | SK_pi | SK_pr = prf+(SKEYSEED, Ni | Nr | SPIi | SPIr)

      – SK_e and SK_a are used for IKE encryption and integrity protection
      – SK_d is used for derivation of further keying material for CHILD_SAs
      – SK_p are used in the input data of AUTH

Initial Exchanges: IKE_SA_INIT and IKE_AUTH (cont.)

- **The initiator and responder assert their identities with the ID payload**
- **AUTH payload is the proof of knowledge of the secret corresponding to ID and is used to integrity protect the contents of the IKE_SA_INIT messages**
- **Optionally, messages 3 and 4 may include a certificate, or certificate chain providing evidence that the key used to compute a digital signature belongs to the name in the ID payload**
- **The optional payloads IDr (for initiator) and IDi (for responder) enable both entities to specify which peer's identities they wants to talk to**
- **The optional payloads TSi and TSr report the proposed traffic selectors (in the 3rd message) and the accepted subset of traffic selectors (in the 4th message)**
  - IP packet "protect" selectors are stored in a Security Policy Database (SPD)

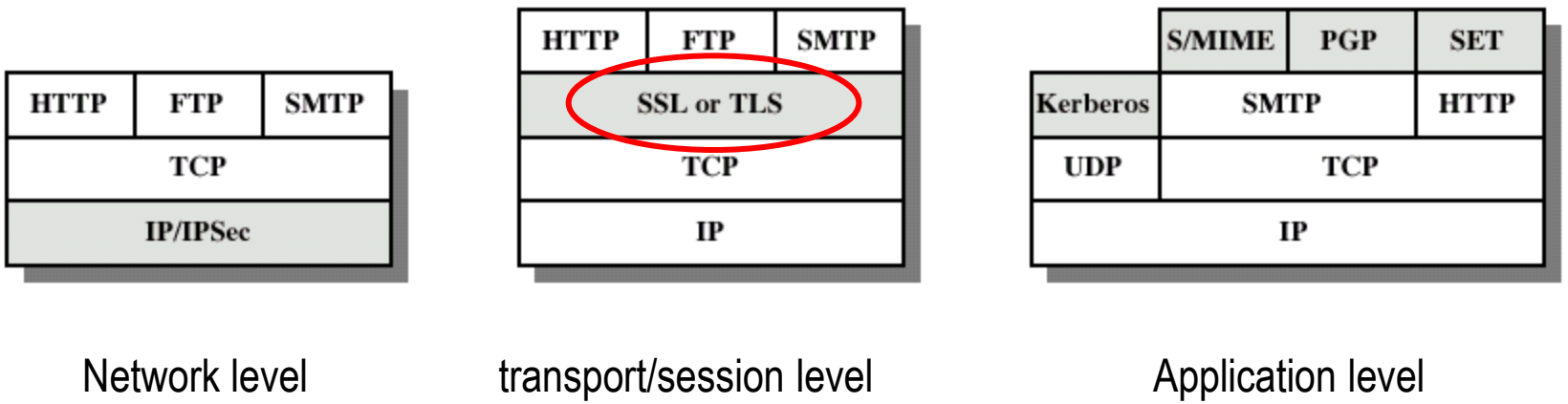
Authentication of the IKE SA

- The signature or MAC will be computed using algorithms dictated by the type of key used by the signer, and specified by the Auth Method field in the Authentication payload
- In case of shared secret
  - **AUTH = prf(prf(Shared Secret,"Key Pad for IKEv2"), <data>)**
    - the pad string is added so that if the shared secret is derived from a password, the IKE implementation can store the value prf(Shared Secret,"Key Pad for IKEv2")
- In case of public-key
  - **RSA or DSA digital signature over the hash value**
- Data to be “signed” start with the first octet of the first SPI in the header and end with the last octet of the last payload
  - **for the responder, appended to this are the initiator's nonce Ni and the value prf(SK_pr,IDr') where IDr' is the responder's ID payload excluding the fixed header**
  - **for the initiator, appended to this are the responder's nonce Nr, and the value prf(SK_pi,IDi')**

TLS and DTLS

Transport Layer Security (TLS)

- Transport Layer Security (TLS) protocol (RFC 5246, version 1.2) provides privacy and data integrity between two communicating client/server applications
- TLS is application protocol independent
  - **higher-level protocols can layer on top of the TLS protocol transparently**



TLS security services

- TLS provides:
  - **peer's identity authentication**
    - using asymmetric cryptography (e.g., RSA, DSA, etc.)
  - **negotiation of encryption algorithm and cryptographic keys**
    - protected against eavesdropping, modifying and replay attacks
  - **confidentiality**
    - symmetric cryptography is used
  - **Message integrity and data authentication**
    - message transport includes a message integrity check using a keyed MAC
    - secure hash functions (e.g., SHA-1, etc.) are used for MAC computations
- TLS uses X.509 server certificates and client certificates (optional)

TLS vs. SSL

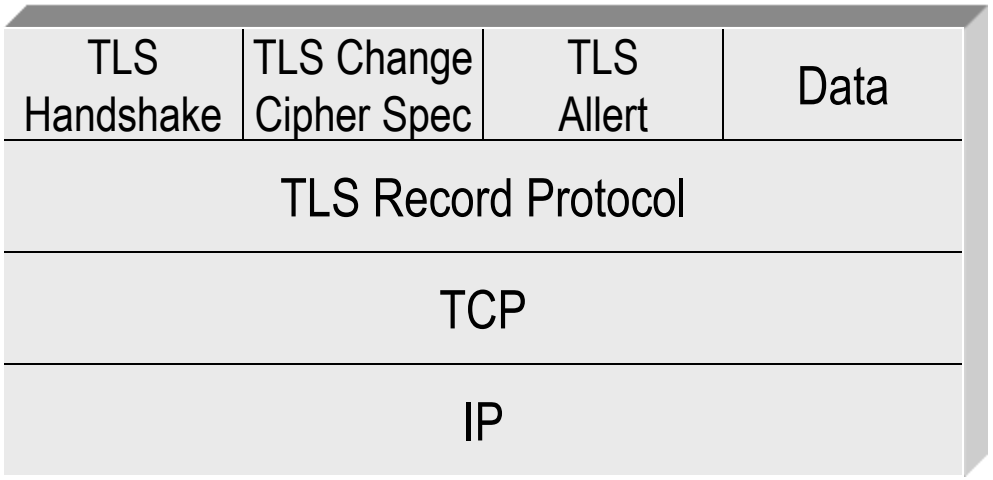
- TLS protocol (IETF standard) was based on the SSL 3.0 Protocol Specification as published by Netscape
- TLS v1.0 == SSL v3.1
  - **the differences between TLS v1.0 and SSL 3.0 were minor:**
    - record format version number
    - uses HMAC for MAC
    - a pseudo-random function expands secrets
    - has additional alert codes
    - some changes in supported ciphers
    - changes in certificate negotiations
    - changes in use of padding

TLS phases

- 1) Handshake
  - **Establish connection**
  - **Agree on encryption algorithm**
  - **Exchange key**
  - **Authentication**
    - through certificates
    - server only authentication, or both client and server
- 2) Securing messages
  - **Sending the actual enchrypted messages**
  - **Integrity checks with MACs**

TLS protocol

- The TLS Protocol is a layered protocol
  - **At each layer, messages may include fields for length, description, and content**
- The protocol is composed of two layers:
  - **TLS Handshake Protocol**
  - **TLS Record Protocol**

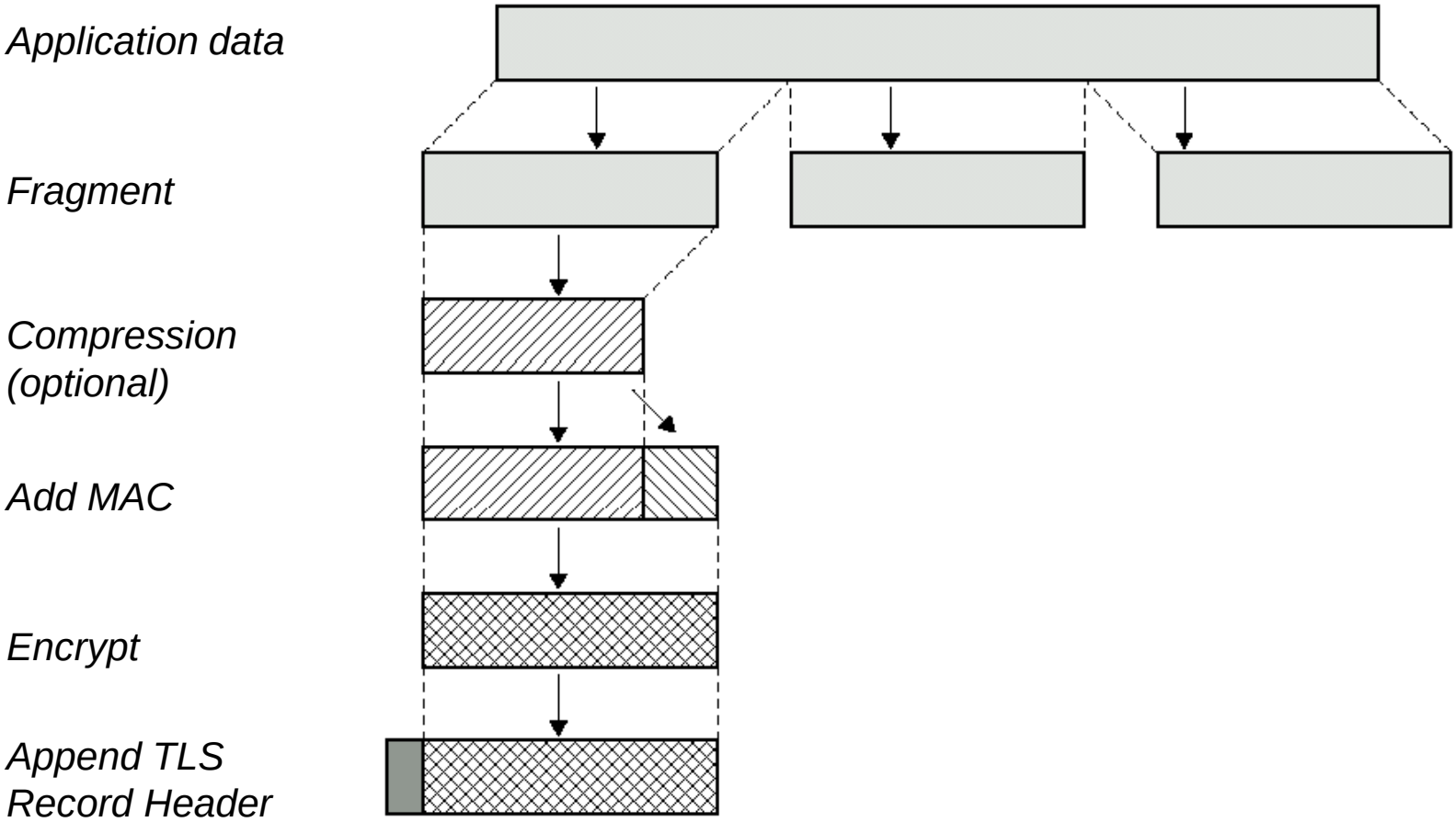


TLS Record Protocol

- TLS Record Protocol is at the lowest level, layered on top of some reliable transport protocol (e.g., TCP)
- Fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, and transmits the result
- Provides connection security:
  - **confidentiality**
    - symmetric cryptography is used for data encryption (e.g., AES, RC4, etc.
    - the keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol)
  - **Message integrity and data authentication**
    - message transport includes a message integrity check using a keyed MAC
    - secure hash functions (e.g., SHA-1, etc.) are used for MAC computations
- It is used for encapsulating higher-level TLS protocols
  - **currently, four protocols are considered: the handshake protocol, the alert protocol, the change cipher spec protocol, and the application data protocol**

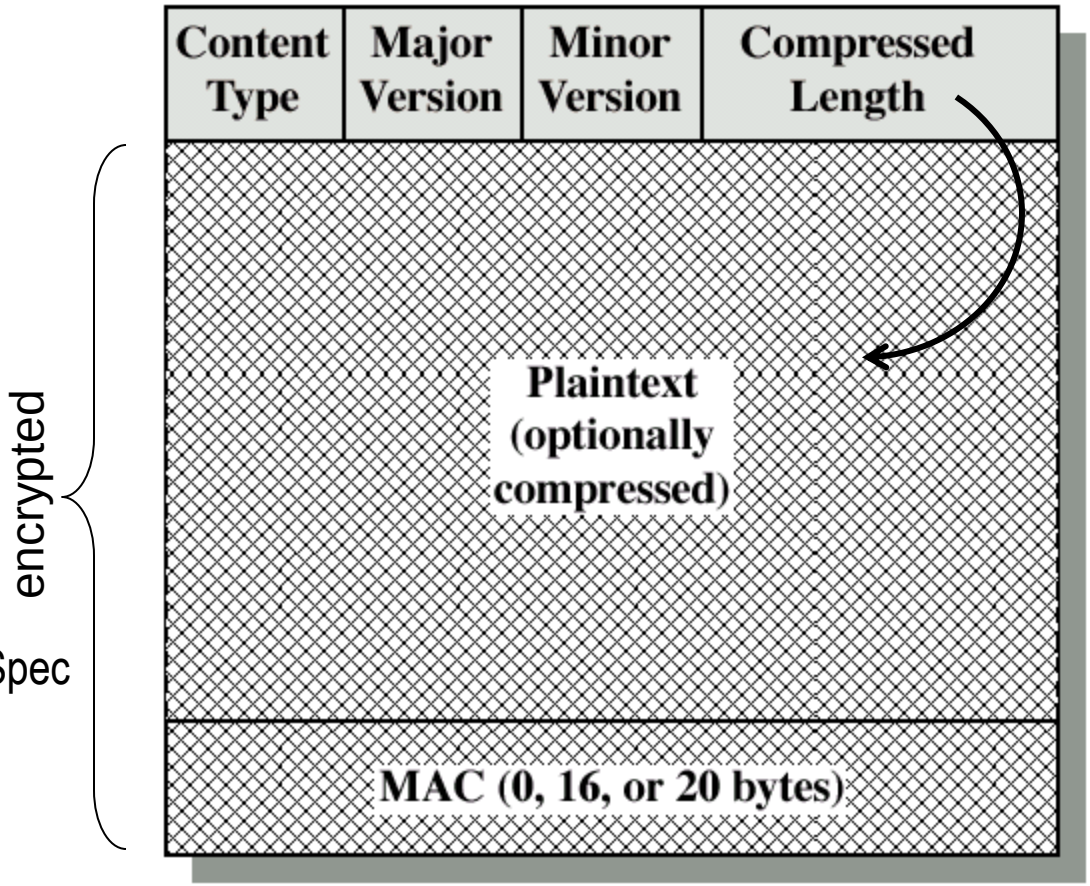
TLS Record Protocol (cont.)

- TLS-RP operation:



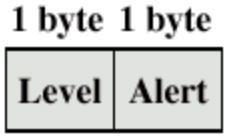
TLS Record Protocol (cont.)

- TLS-RP format
  - **Major Version=3**
  - **Minor Version:**
    - 0 = SSL3.0
    - 1 = TLS1.0
    - 2 = TLS1.1
    - 3 = TLS1.2
    - 4 = TLS1.3
  - **Content-type:**
    - 20 = Change Cihper Spec
    - 21 = Alert
    - 22 = Handshake
    - 23 = Data



TLS Alert Protocol

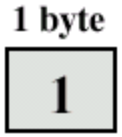
- Alert messages convey the severity of the message (warning or fatal) and a description of the alert
- Message consists of two bytes (Level type and Alert code)



- Two levels of alerts are defined
  - **warning (first byte=1)**
    - close notify, no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown
  - **fatal (first byte=2)**
    - unexpected message, bad record mac, decompression failure, handshake failure, illegal parameter
    - result in the immediate termination of the connection
- Like other messages, alert messages are encrypted
- TLS-RP Content-type 21

TLS Change Cipher Spec Protocol

- The change cipher spec protocol exists to signal transitions in ciphering strategies
- The protocol consists of a single message, which is encrypted and compressed under the current (not the pending) connection state
- ChangeCipherSpec message consists of a single byte of value 1



- The message is sent by both the client and the server to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys
- TLS-RP Content-type 20

TLS Handshake Protocol

- TLS Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data
- The TLS Handshake Protocol provides:
  - **authentication of the the peer's identity using asymmetric cryptography (e.g., RSA, DSA, etc.)**
  - **secure negotiation of a shared secret**
- TLS-RP Content-type 22

Original cert-based key exchanges

- DHE-RSA and ECDHE-RSA
  - **DH exchange signed using RSA keys**
  - **forward secrecy**
- DHE_DSS
  - **DH exchange signed using DSS keys**
  - **forward secrecy**
- RSA
  - **key exchanged encrypted by the client with the RSA key of the server**
  - **has been removed in TLS 1.3**
- DH_RSA, ECDH-RSA, DH_DSS, and ECDH-ECDSA
  - **use static DH cert (signed with RSA/DSS)**
  - **have been removed in TLS 1.3**

TLS 1.3 key exchanges

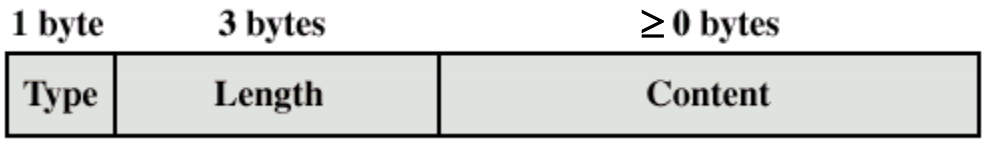
- TLS 1.3 supports three basic key exchange modes:
  - **(EC)DHE**
    - Diffie-Hellman over either finite fields or elliptic curves
  - **PSK-only**
  - **PSK with (EC)DHE**

TLS Handshake Protocol (cont.)

- When a TLS client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets
- The TLS Handshake Protocol involves the following steps:
  - **exchange hello messages to agree on algorithms, exchange random values, and check for session resumption**
  - **exchange the necessary cryptographic parameters to allow the client and server to agree on a pre-master secret**
  - **exchange certificates and cryptographic information to allow the client and server to authenticate themselves**
  - **generate a master secret from the premaster secret and exchanged random values**
  - **provide security parameters to the record layer**
  - **allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker**

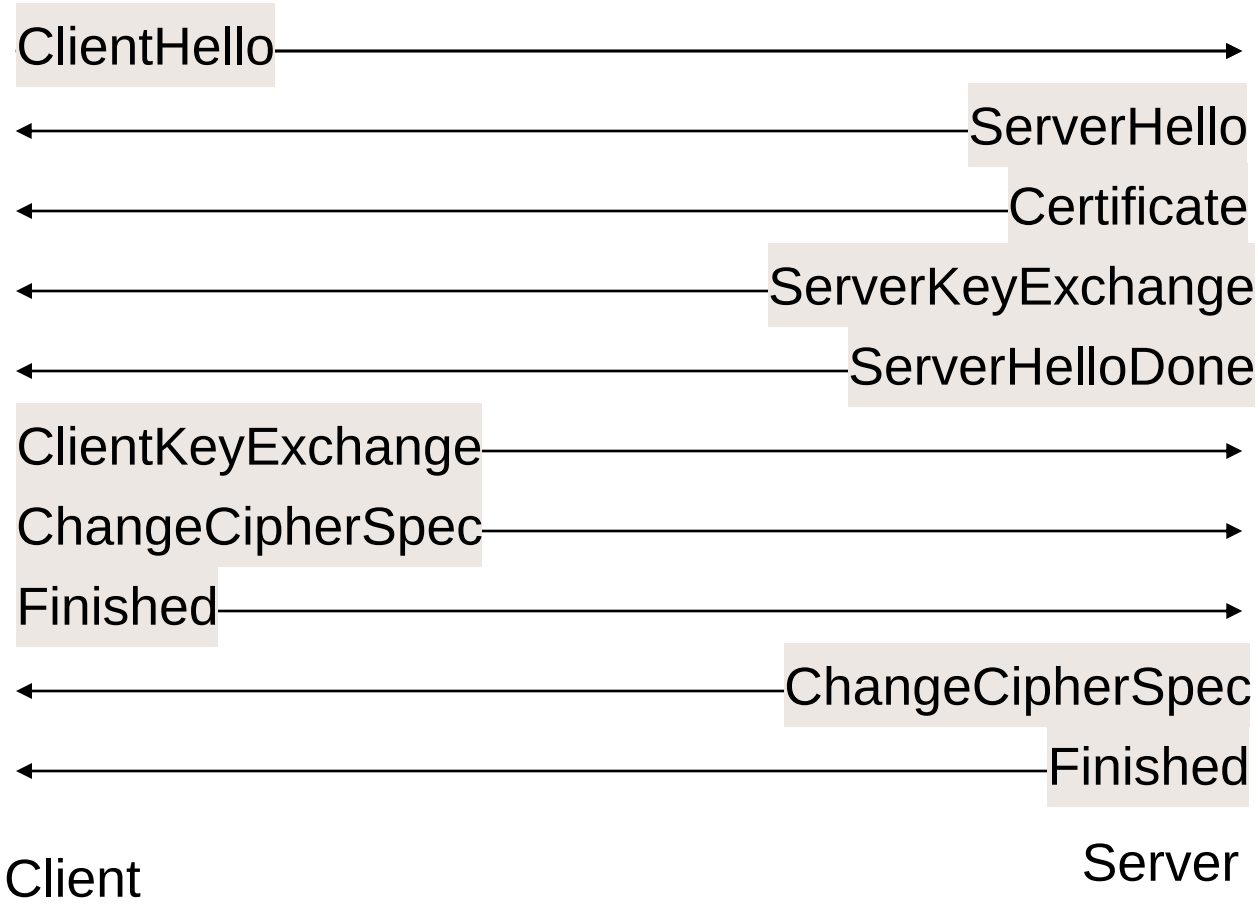
TLS Handshake Protocol (cont.)

- TLS Handshake Protocol message format:



- Currently defined TLS Handshake Protocol message types:
  - **client_hello (1)**
  - **server_hello (2)**
  - **certificate (11)**
  - **server_key_exchange (12)**
  - **certificate_request (13)**
  - **server_hello_done (14)**
  - **certificate_verify (15)**
  - **client_key_exchange (16)**
  - **finished (20)**

TLS 1.2 Handshake Protocol



TLS Handshake Protocol (cont.)

- ClientHello
  - **the client starts the communication by sending the ClientHello message**
  - **contains**
    - version number
    - random
    - optional session ID
      - used to resume a previous session
    - list of cipher suites supported
      - The cipher suite includes key exchange algorithm, symmetric algorithm (including chaining mode) and MAC algorithm

TLS Handshake Protocol (cont.)

- ServerHello
  - **sent in response to the ClientHello message**
  - **with this message, the server finally decides which cipher suite to use**
  - **contains**
    - version number
    - random
    - optional session ID (for resuming a previous session)
    - the cipher suite to be used, picked from the list of proposals given by the client
- Certificate
  - **contains the server certificate, including the chain leading up to the CA root certificate**
    - Optional according to the TLS specifications, but most (all?) implementations require a server certificate
    - If no certificate is sent, the ServerKeyExchange is required

TLS Handshake Protocol (cont.)

- ServerKeyExchange
  - **used for the key exchange**
    - Includes the server part of the key exchange
    - Exact meaning depends on the cipher suite chosen
      - For Diffie-Hellman, the modulus p, the generator g and $y = g^x$ is sent
    - Only when the server Certificate message (if sent) does not allow the client to generate a premaster secret
      - This is true for the following key exchange methods:
        - » DHE_DSS
        - » DHE_RSA
        - » DH_anon (removed)
- ServerHelloDone
  - **marks the end of the server's part in the handshake**
    - It does not contain any other information

TLS Handshake Protocol (cont.)

- ClientKeyExchange
  - **always sent by the client**
  - **contains the client part in the key agreement**
    - the premaster secret is set, either
      - by direct transmission of the RSA-encrypted secret or
      - By the transmission of DH parameters
  - **the exact format depends on the exchange algorithm agreed on previously**
    - for Diffie-Hellman, the message contains the client's part $y = g^x \bmod p$
- ChangeCipherSpec
  - **indicates that from this point, communication is encrypted**
- Finished
  - **is encrypted**
  - **marks the end of the handshake**

TLS Handshake Protocol (cont.)

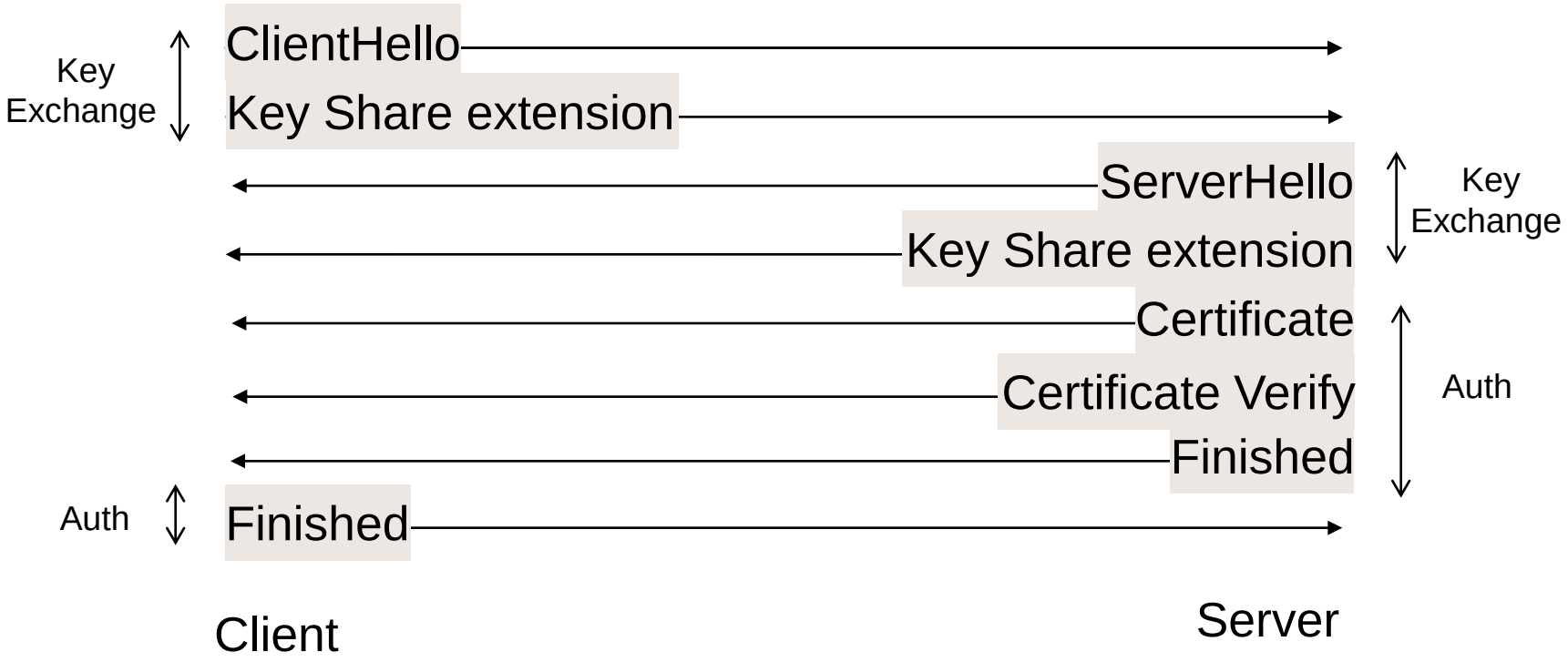
- ChangeCipherSpec and Finished
  - **play the same role as the client's message**

- After the handshake is complete, the client and the server start exchanging encrypted messages

Computing the Master Secret

- RSA
  - **48-byte pre_master_secret is generated by the client, encrypted under the server's public key, and sent to the server**
- DH or DHE
  - **A conventional Diffie-Hellman computation is performed**
    - The negotiated DH key is used as the PMSK

TLS 1.3 Handshake Protocol



TLS 1.3 Handshake Protocol

- In the Key Exchange phase
  - **client sends the ClientHello message**
    - random nonce, offered protocol versions, a list of symmetric ciphers
    - either a set of Diffie-Hellman key shares (in the "key_share" extension), a set of pre-shared key labels (in the "pre_shared_key" extension), or both
  - **server processes the ClientHello, determines the appropriate cryptographic parameters, and responds with ServerHello**
- The combination of the ClientHello and the ServerHello determines the shared keys
  - **if (EC)DHE key establishment is used, the ClientHello and ServerHello contain the ephemeral Diffie-Hellman values**
- In the Authentication phase
  - **in case of certificate-based authentication, the server sends its certificate and signature over the entire handshake using the private key**
  - **Finished messages contain a MAC over the entire handshake**
    - provide key confirmation, and bind the endpoint's identity to the exchanged keys

Generating secretes

- Key exchange methods establish a “Pre Master Secret” (PMSK) between the client and server
- For all key exchange methods, the same algorithm is used to convert the PMSK into the “Master Secret” (MSK) [48B]
  - **the PMSK should be deleted from memory once the MSK has been computed**

MSK = PRF(PMSK, "master secret", ClientHello.random || ServerHello.random)

  - **PRF:**
    - PRF(secret, label, seed) = P_hash(secret, label || seed)
    - P_hash(secret, data) is a data expansion function defined for expand a secret and seed into an arbitrary quantity of output:
      - P_hash(secret, data) = H_1 || H_2 || H_3 || ..
        - » H_i = HMAC_hash(secret, A_i || data)
        - » A_0 = data
        - » A_i = HMAC_hash(secret, A_{i-1})
    - P_hash can be iterated as many times as necessary to produce the required quantity of data

Generating secretes (cont.)

- MSK is used to generates secrete material such as
  - **a client write MAC secret,**
  - **a server write MAC secret,**
  - **a client write key,**
  - **a server write key,**
  - **a client write IV, and**
  - **a server write IV**
- MSK is hashed into a sequence of secure bytes

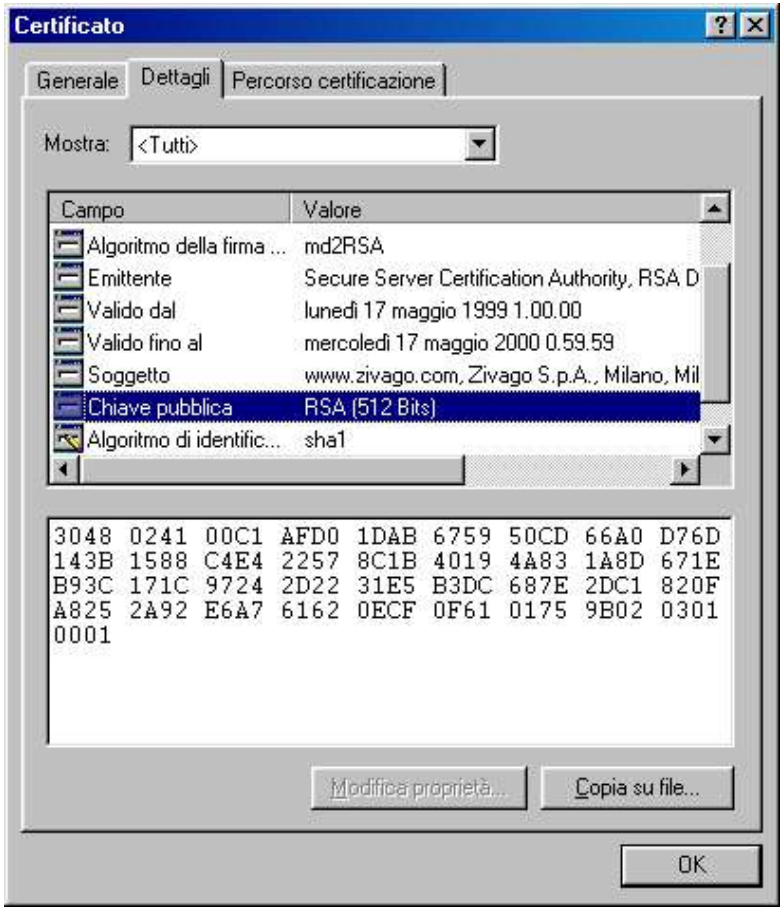
  key_block = PRF(MSK, "key expansion", server_random || client_random)
- assigned to the above secret material
  - **MAC secrets, encryption keys, IVs...**

Verifying the certificate

- All certificates in TLS are in the X.509 format
- To verify that a certificate is valid, the verifier must
  - **Check that the CA signature is valid**
  - **Check that the owner of the certificate knows the private key**
  - **Check that the identifying information is what it should be**
- The protocol specifies how to perform the first two parts, but the last part is up to the implementation
- The CN field of the server certificate contains the host name of the server

Web Browser Support

- Some Root/CA certificates pre-installed
  - **Firefox Opera and IE have different lists**
  - **CA's public key is used to verify the signatures on issued certificates**
  - **Browsers can accept unverifiable certificates or alert the user**
- Users can install additional certificates
  - **Additional Root/CA certificates**
    - Security vulnerability
  - **Client certificates**



Datagram Transport Layer Security (DTLS)

- TLS cannot be directly used with UDP, since packets may be lost or reordered (unreliability)
  - **TLS does not allow independent decryption of individual records**
    - because the integrity check depends on the sequence number, if record N is not received, then the integrity check on record N+1 will be based on the wrong sequence number and thus will fail
    - prior to TLS 1.1, there was no explicit IV and so decryption would also fail
  - **TLS handshake layer assumes that handshake messages are delivered reliably**
    - breaks if those messages are lost
- Datagram Transport Layer Security (DTLS) Version 1.2
  - **TLS over datagram transport (UDP)**
  - **DTLS makes only the minimal changes to TLS required to fix the problem**

DTLS (cont.)

- Loss-Insensitive Messaging
  - **In TLS Record Layer records are not independent**
    - Cryptographic context of stream cipher key stream is retained between records
    - Anti-replay and message reordering protection are provided by a MAC that includes a sequence number, but the sequence numbers are implicit in the records
  - **DTLS solves the first problem by banning stream ciphers**
  - **DTLS solves the second problem by adding explicit sequence numbers**

DTLS (cont.)

- Providing Reliability for Handshake
  - **the TLS handshake is a lockstep cryptographic handshake**
    - messages must be transmitted and received in a defined order; any other order is an error
      - this is incompatible with reordering and message loss
  - **DTLS uses a simple retransmission timer to handle packet loss**
  - **in DTLS, each handshake message is assigned a specific sequence number within that handshake**
    - when a peer receives a handshake message, it can quickly determine whether that message is the next message it expects
- TLS handshake messages are potentially larger than any given datagram, thus creating the problem of IP fragmentation
  - **In order to compensate for this limitation, each DTLS handshake message may be fragmented over several DTLS records, each of which is intended to fit in a single IP datagram**
    - each DTLS handshake message contains both a fragment offset and a fragment length